POLITÉCNICO DE LEIRIA

Computer Graphics

2222148 - Catarina Veloso

2221903 - João Vareta

Project development

Our strategy for this project was to use the base that we had from class to do the 2D rendering of the sprite animations. We created a Texture class that is responsible for image loading, and from then, we worked on the animation system, where the program goes through the spritesheets of the characters and picks them accordingly to be displayed in the scene. To work with multiple objects, we tried multiple options, such as handling the texture files with a single array, binding them in the render loop with another loop, and doing them individually.

Animation

For the animation of each sprite sheet, we created a function that would divide a singular sheet into whatever many sprites existed in said sheet. We can insert how many sprites per row and column on the function. Then, in the while function, the method will run through each divided part of the sheet and run through it at 12 frames per second, effectively animating the object in the sheet.

Multiple Object Rendering

For rendering multiple objects at once, we tried multiple solutions. As stated before, the solution we settled for was taking care of each object individually. We thought about making specific methods on an external class to help with this, but due to time constraints, we decided that it would be best if we had working code that could be duplicated any time we wanted a new object.

For an object to be displayed in the scene, you need 4 new vertices per object, which represent the corners of the sprite. After defining these vertices, you then need to load the desired texture and define a new variable that will store it. We chose the sprite names for simplicity’s sake. After this, in the render loop, we bind each texture to their respective set of 4 vertices and draw them.

Vertical Scrolling

To give the idea of vertical scrolling, we decided to add background rocks moving downwards. We initially had trouble with it since we had to render multiple sections of a spritesheet at the same time. Since our code was made to only pick a single sprite index from an entire spritesheet, we had to improvise and adapt the spritesheet size settings code to accommodate for a small rock formation that was on the spritesheet.

After setting up the correct requirements, we simply made the vertice group of the rock texture move downwards, and reset to a position off screen to then scroll back down one more time, doing this indefinitely on a loop.

We then mirrored the rock to the other side and gave it an offset so that the background becomes more dynamic.

Enemy Movement

For the enemy movement, we settled on a very simple solution, which was to make one of them move from side to side. We used the same strategy as the rock formations for movement, but instead of using the logic that moved the rock offscreen to reset it, we shifted the speed value of the enemy so that it moved in a different direction once it reached a certain point in its path. It also does this indefinitely on a loop.

Ship Movement

For the ship movement, we simply move the vertices bound to the ship texture according to what key is pressed on the keyboard (WASD) to give the idea that the ship is moving around. The strategy for moving it was similar to the enemy or the rock formations, but instead of autonomously moving itself bound by rules, it awaits user input and acts accordingly.

UI

For the UI, we used the same logic from the Animation process and used it to render individual letters. We would insert a number that would be assigned to a specific letter, which would be picked up in the grid we created inside the spritesheet, and then projected onto the project window. For positioning, they were individually positioned by using glm::translate, but were all sized in the same matrix for consistency.

Sources

Class documentation and project bases provided by the teacher.

Copilot aid for syntax related problems and general bug fixing.